浙江大学

本科实验报告

课程名称：操作系统

姓　　名：

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：

指导教师：李环

2024 年 9 月 14 日

浙江大学操作系统实验报告

实验名称：GDB+QEMU 调试 64 位 RISC-V LINUX

电子邮件地址：_____手机：_____

实验地点：曹西 503 实验日期：2024 年 9 月 14 日

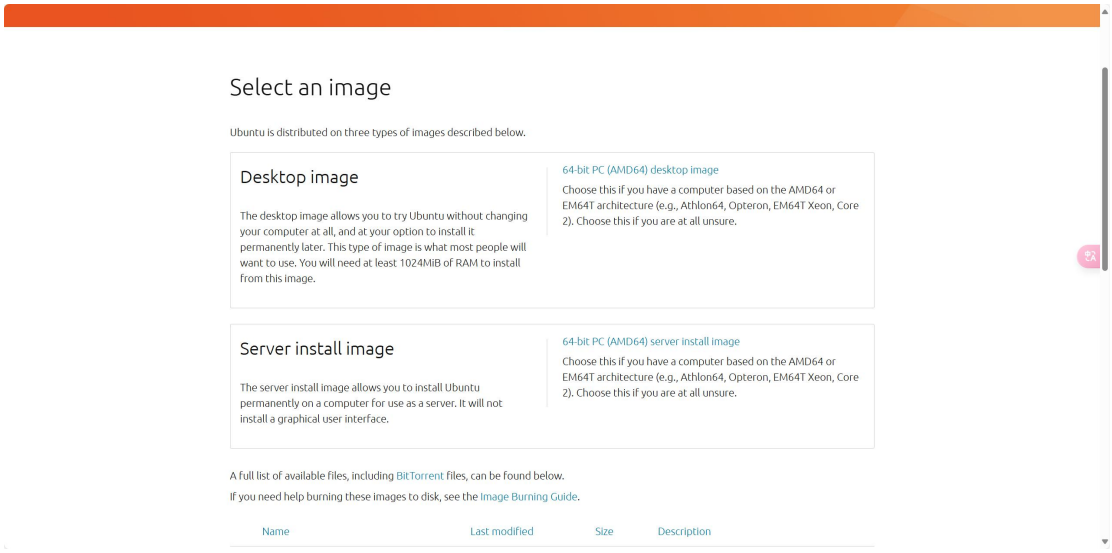
一、实验目的和要求

1. 使用交叉编译工具，完成 Linux 内核代码编译
2. 使用 QEMU 运行内核
3. 熟悉 GDB 和 QEMU 联合调试

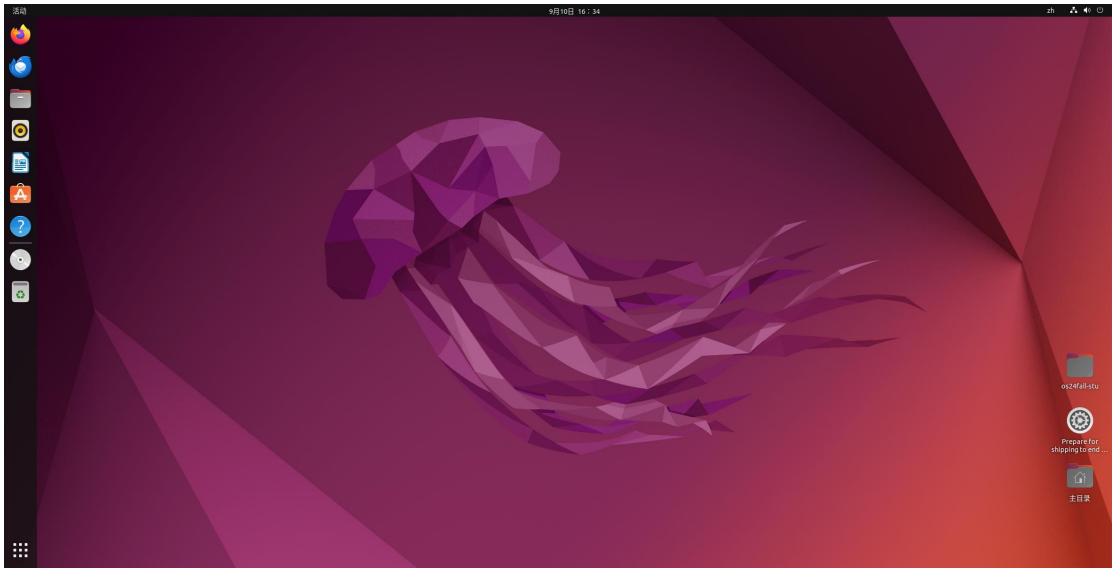
二、实验过程

1. 搭建 Ubuntu 虚拟机环境

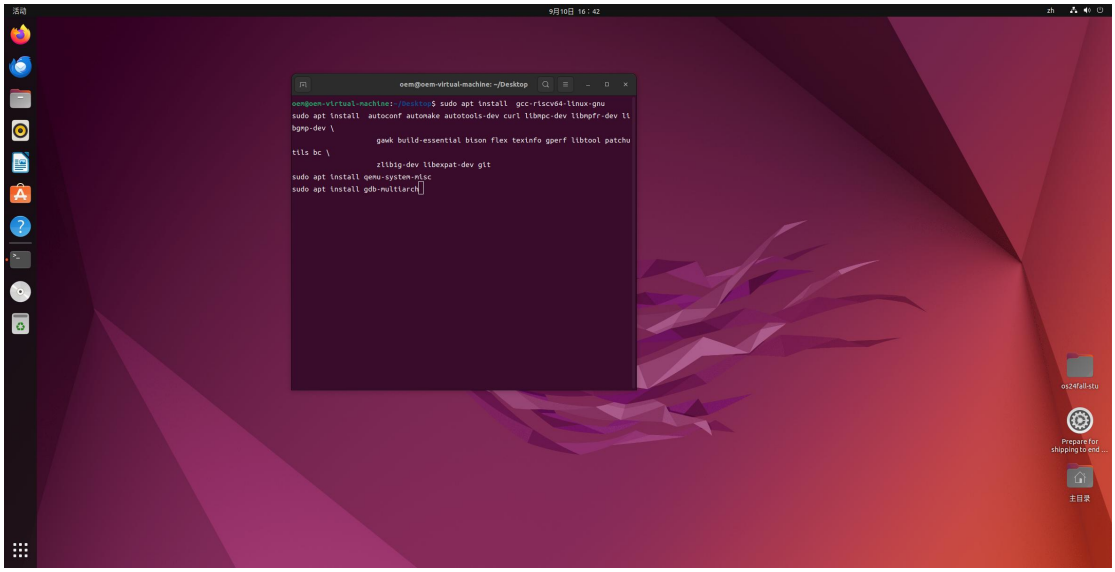
   ①下载 Ubuntu 镜像文件，此处我选择了 https://releases.ubuntu.com/jammy/ 上 22.04 版本的镜像文件



②将镜像文件导入虚拟机中，初步构建 Ubuntu 环境。



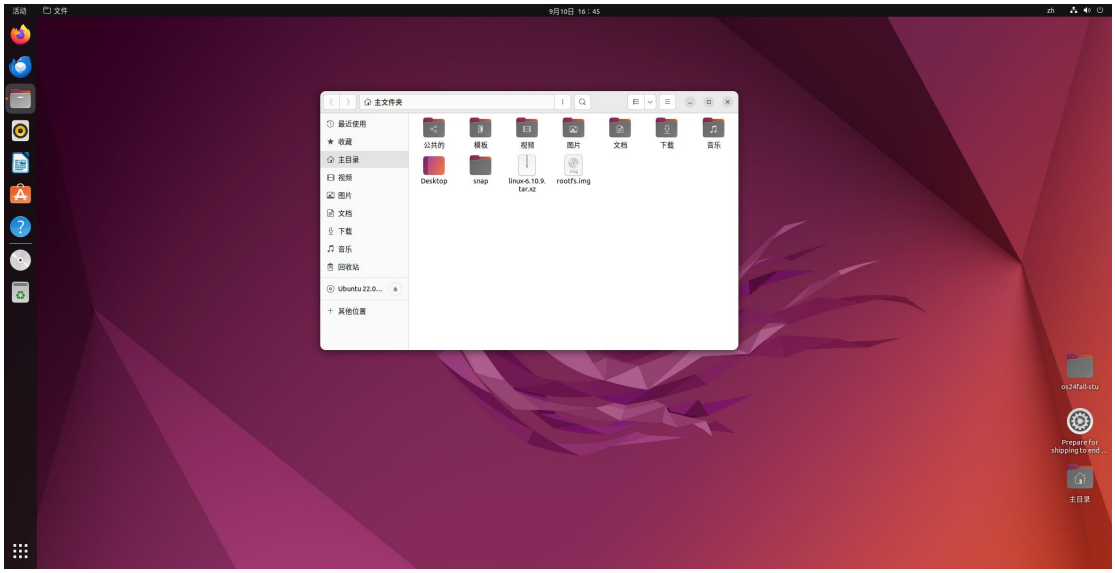
③下载必要的交叉编译工具链、软件包、QEMU 模拟器以及 gdb。



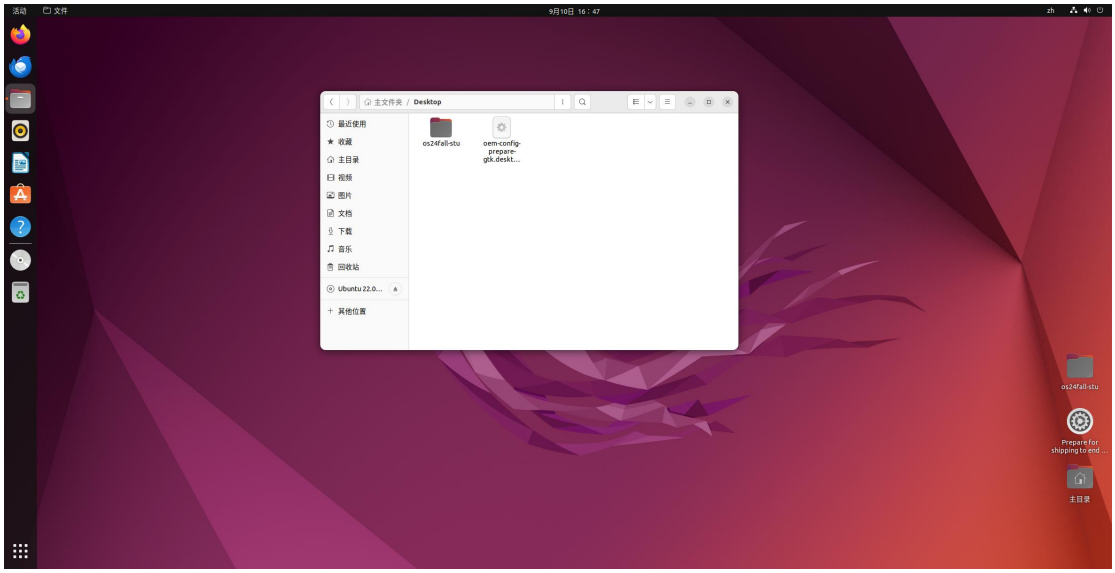
由于事先已经将需要的工具包都安装完成了，这里就不再运行一遍。

2. 获取 Linux 源码和已经编译好的文件系统

①从 https://www.kernel.org 下载 Linux 源码



②使用 git 工具 clone 课程仓库

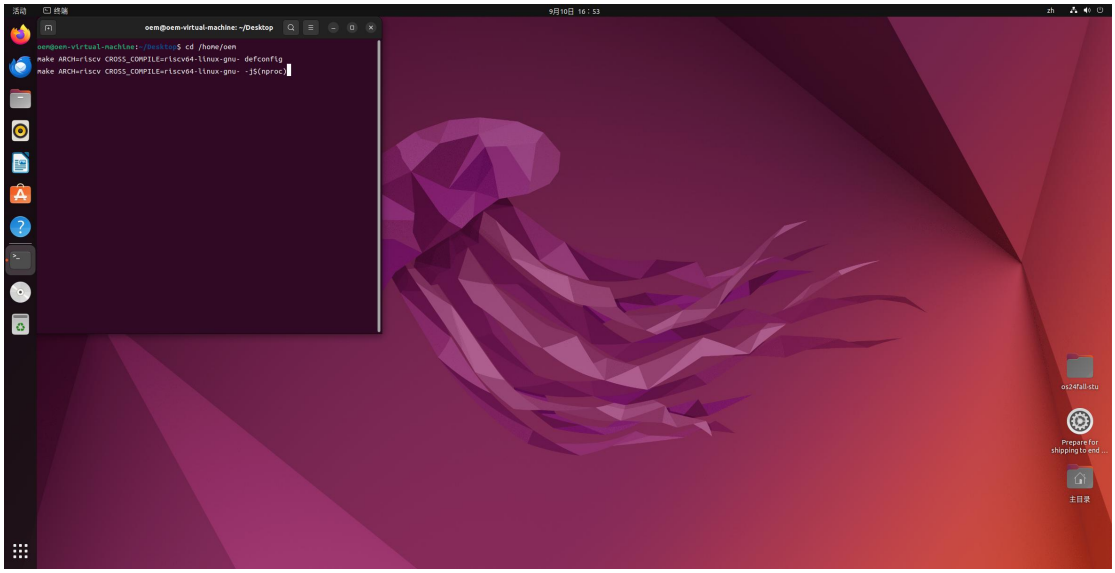


3. 编译 Linux 内核

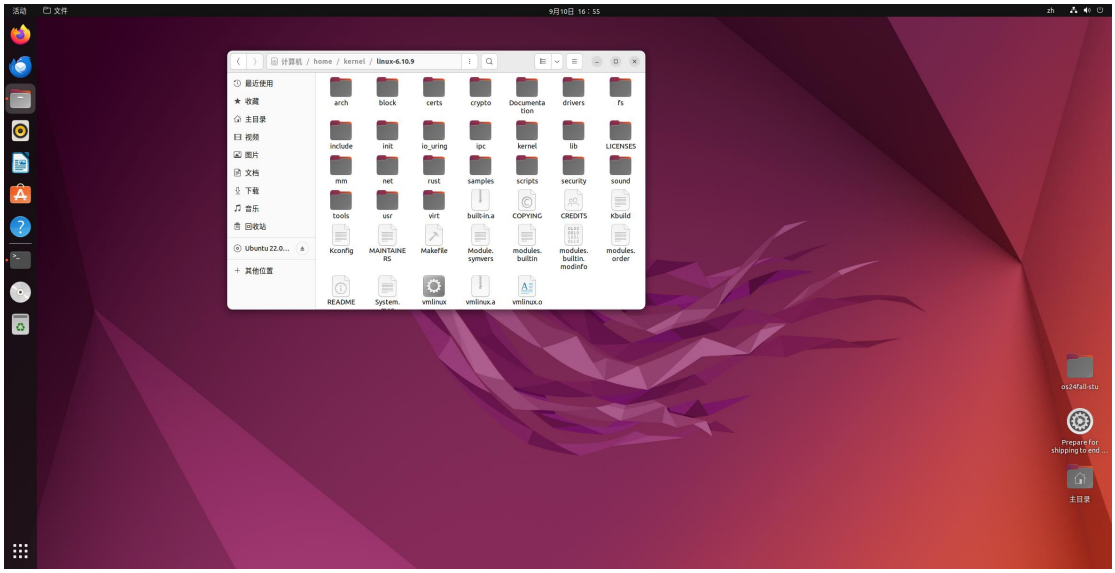
①进入内核源码压缩包的路径。

②使用 xvJf 指令解压压缩包。

③使用交叉编译工具编译 linux kernel，直至编译完成。

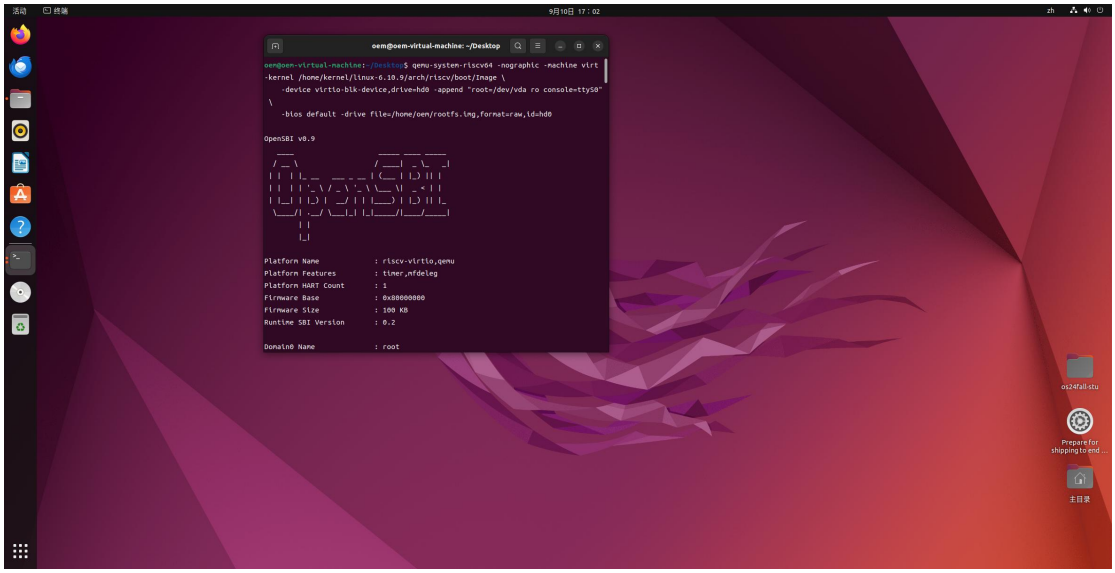


编译完成后的结果：



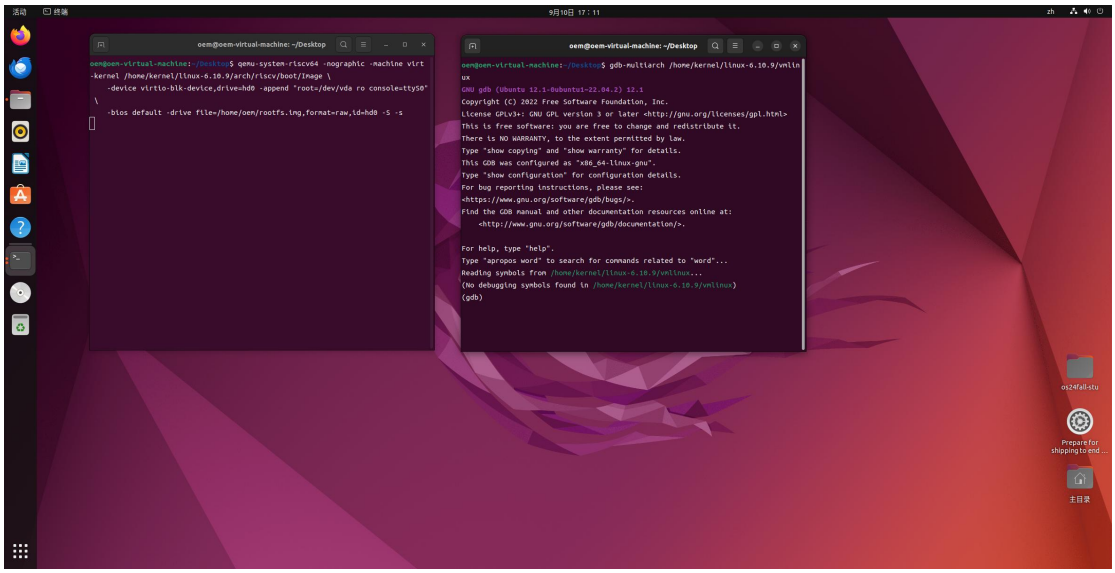
4. 使用 QEMU 运行内核

①使用 qemu 命令启动内核。

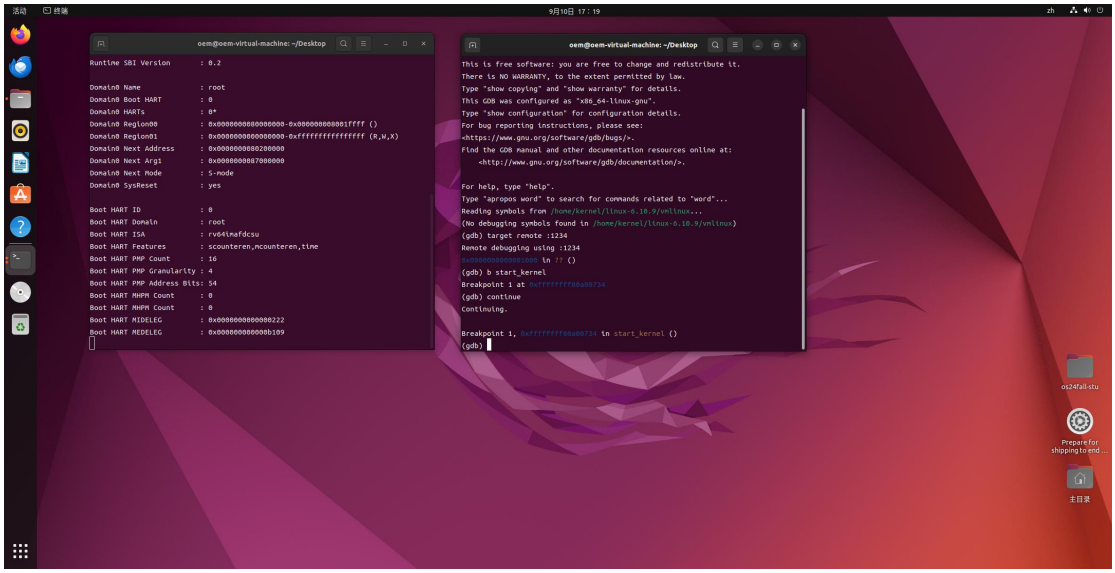


5. 使用 GDB 对内核进行调试

①开启两个 Terminal Session，一个 Terminal 使用 QEMU 启动 Linux，另一个 Terminal 使用 GDB 与 QEMU 远程通信（使用 tcp::1234 端口）进行调试。



②使用 gdb 连接端口 1234，设置断点，continue，让 linux kernel 启动，可以从终端看见，程序在断点处停止。



三、讨论和心得

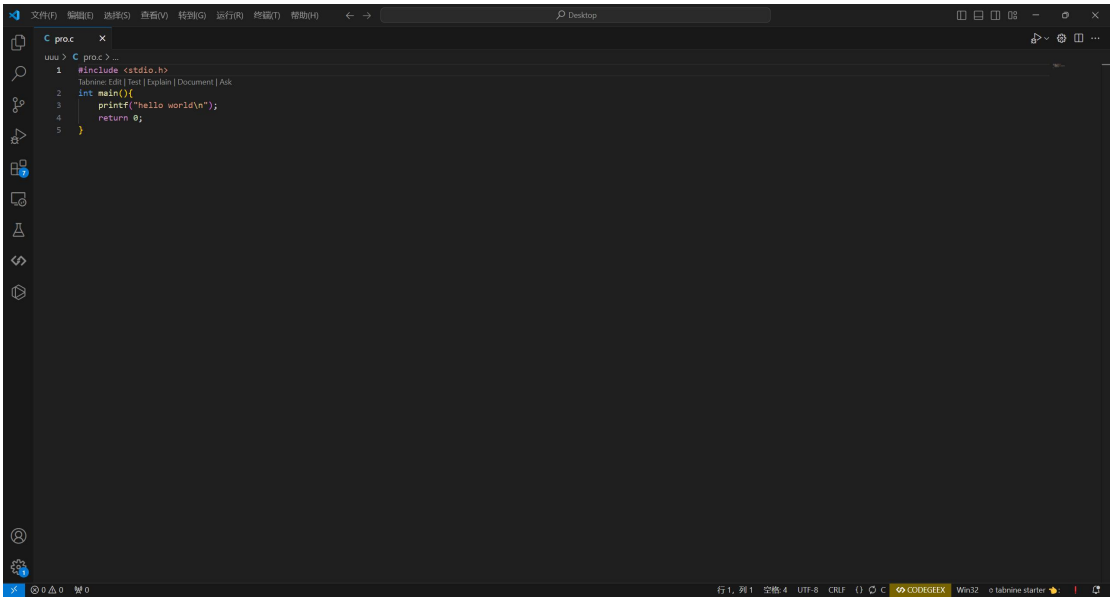
虽然说这次实验总体下来还可以，但是中间仍遇到了不少难题，查阅了很多资料才解决。一开始我是用 ubuntu 的 WSL 系统实验。首先就是 github 克隆仓库的问题，在实验中总能遇到连接 github 网站超时的问题，需要输入 sudo gedit /etc/hosts，在 hosts 文件中加入 xxx.xxx.xxx.xxx github.com git yyy.yyy.yyy.yyy github.com 替换 github 的 ip 地址才能连上，但是实际输入过程中又会出现 gedit 无法支持 display。于是我就把环境放到虚拟机里。前面都很顺利，但是到了编译 linux 内核，文档中有这样一段代码 cd path/to/linux，我一开始以为是直接定位到下载的后缀名为 tar.xz 的路径中，但是这样怎么都编译不成功，后面查阅了资料才发现要先将压缩包解压到某一个路径才能进行编译。最后就是 riscv64-linux-gnu-gcc 和 riscv64-linux-gnu-objdump 的问题，这两段代码的语法都需要自己去查阅资料，在实际运行过程中也遇到了语法错误等问题。最后也是成功完成了整个实验。

这次实验让我了解到了 risc-v linux 系统以及如何用 qemu 和 gdb 联合调试，算是一次比较新鲜的实验，同时也提高了我在陌生操作环境下的操作能力和代码能力，总的来说收获还是很大。

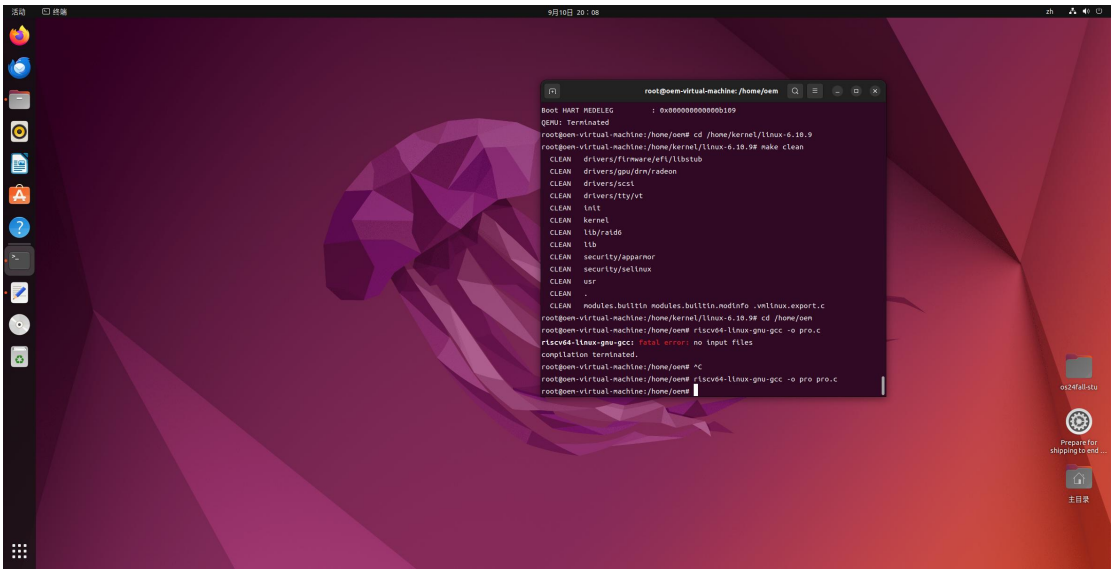
四、思考题

1. 使用 riscv64-linux-gnu-gcc 编译单个.c 文件

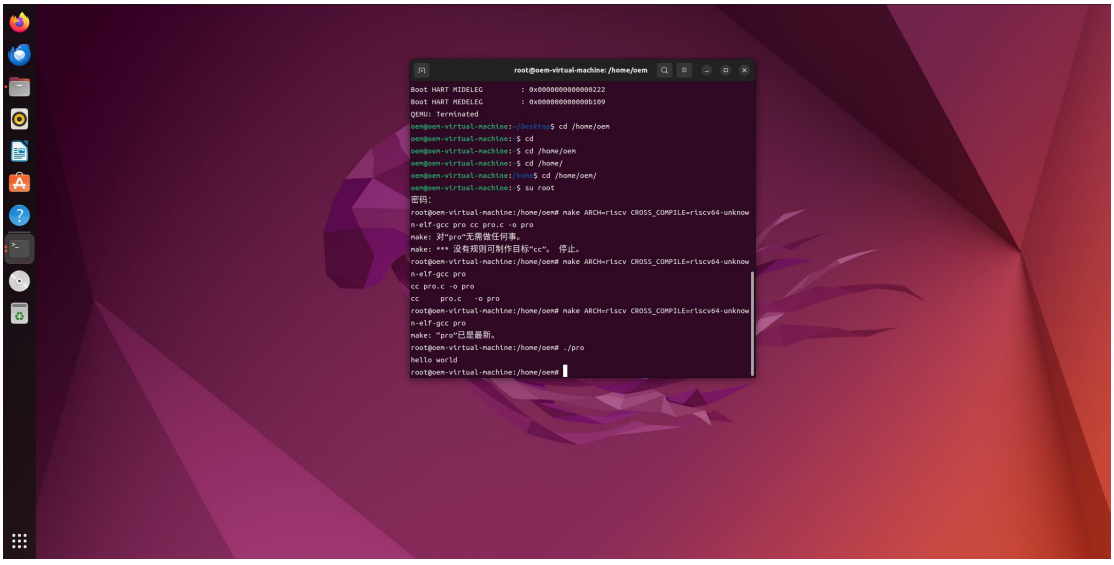
①编写程序如下：



②在终端中采用交叉编译，键入命令如下：



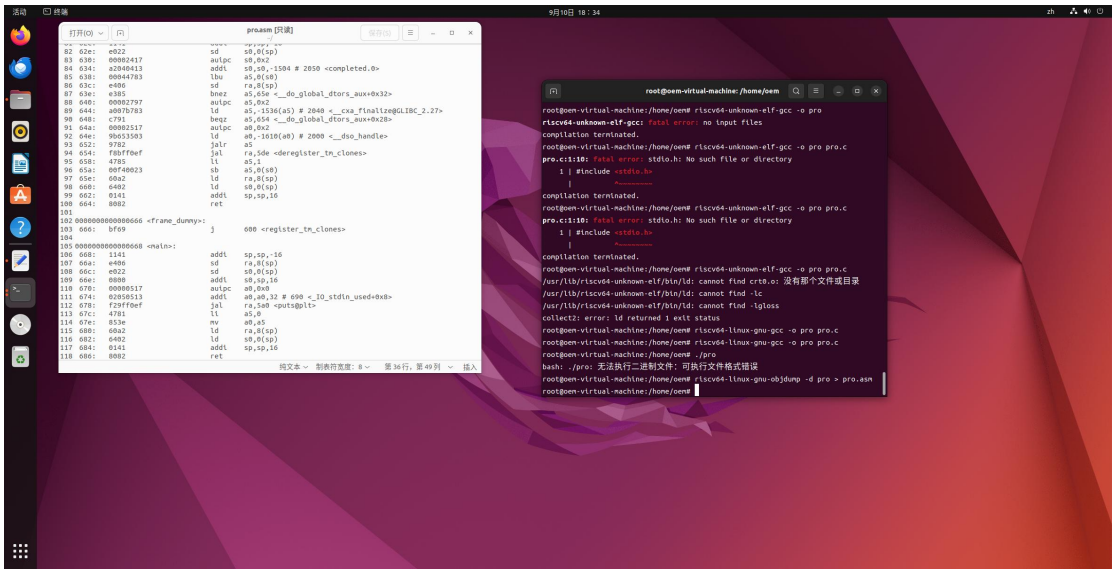
③键入命令后程序运行结果正常。



2. 使用 riscv64-linux-gnu-objdump 反汇编 1 中得到的编译产物

①使用 riscv64-linux-gnu-objdump 进行反编译。

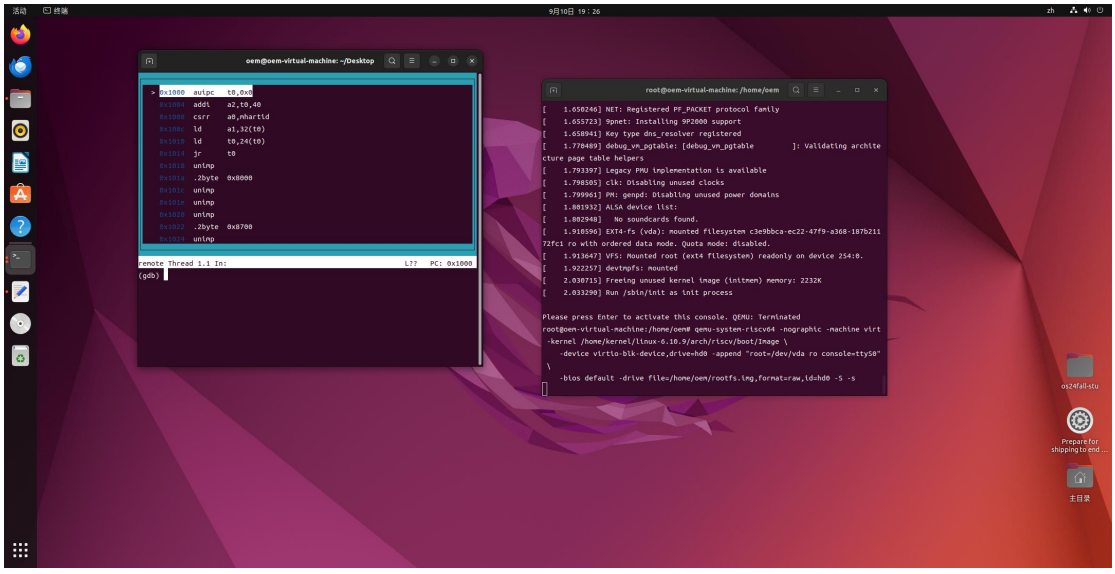
②可以查看生成的文件，确保结果正确。



3. 调试 linux 时：

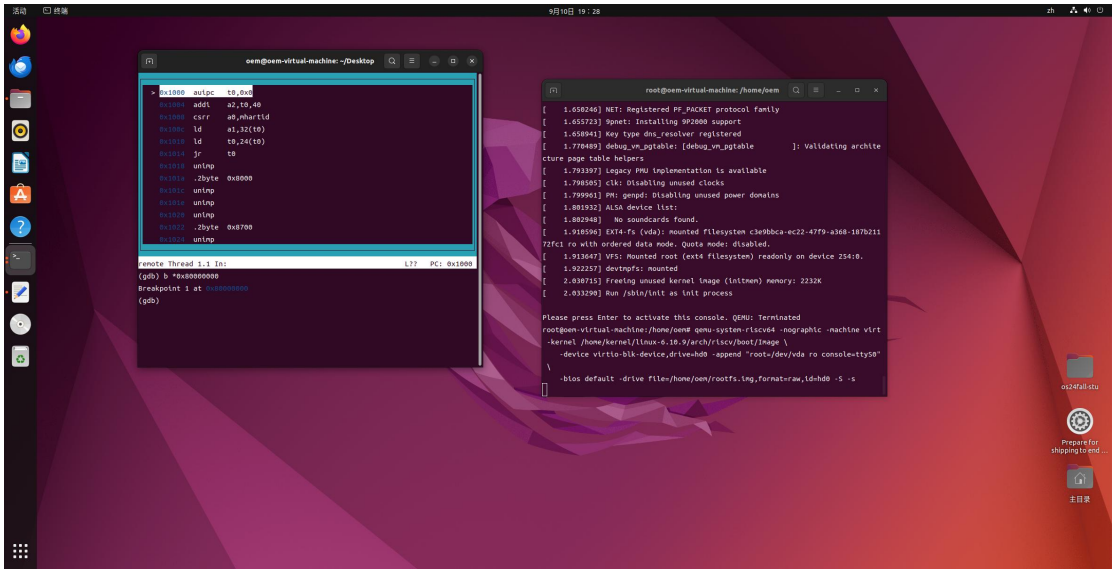
①在 GDB 中查看汇编代码。

在 gdb 中输入 layout asm 即可。



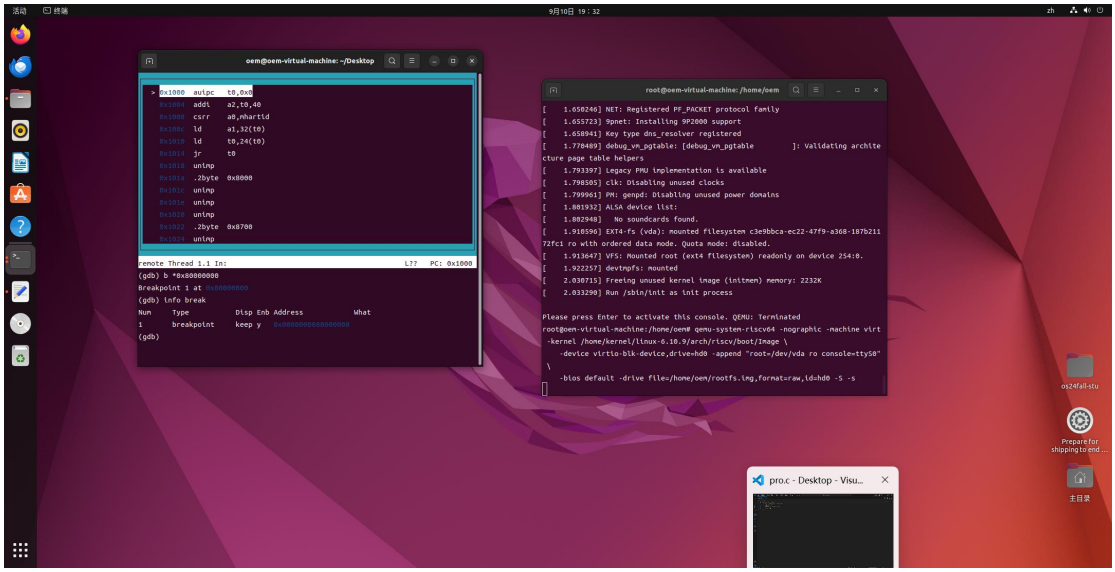
②在 0x80000000 处下断点。

直接输入 b *0x80000000 即可。



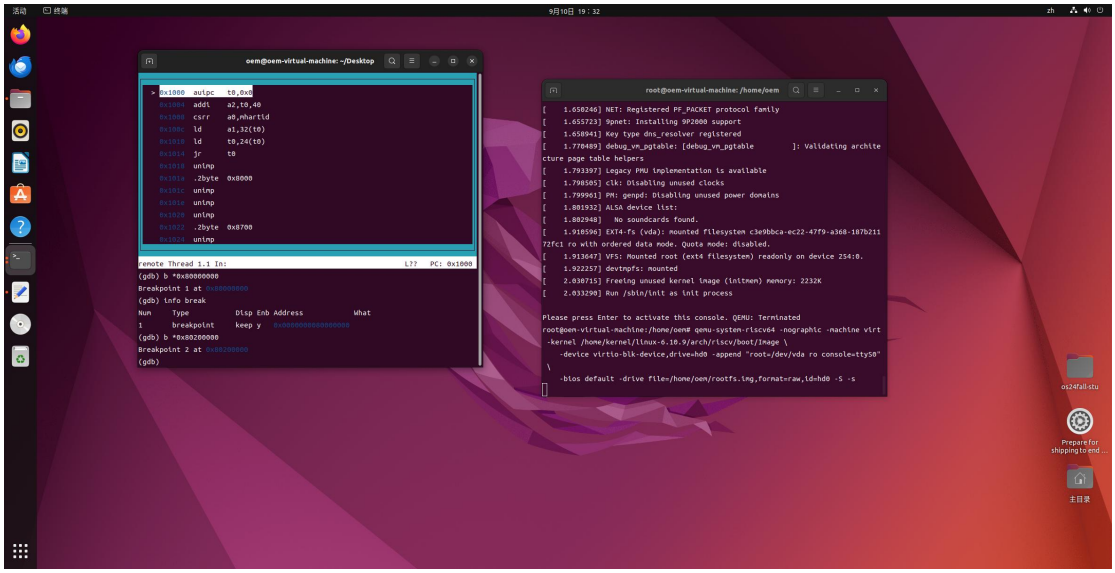
③查看所有已下的断点。

直接输入 info break 即可。



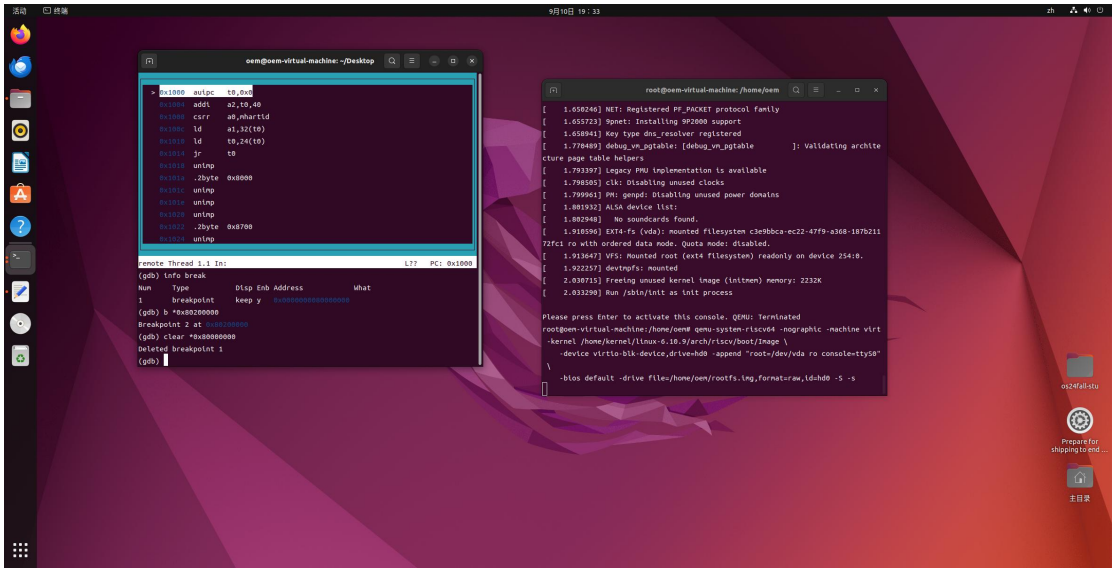
④在 0x80200000 处下断点。

直接输入 b *0x80200000 即可。



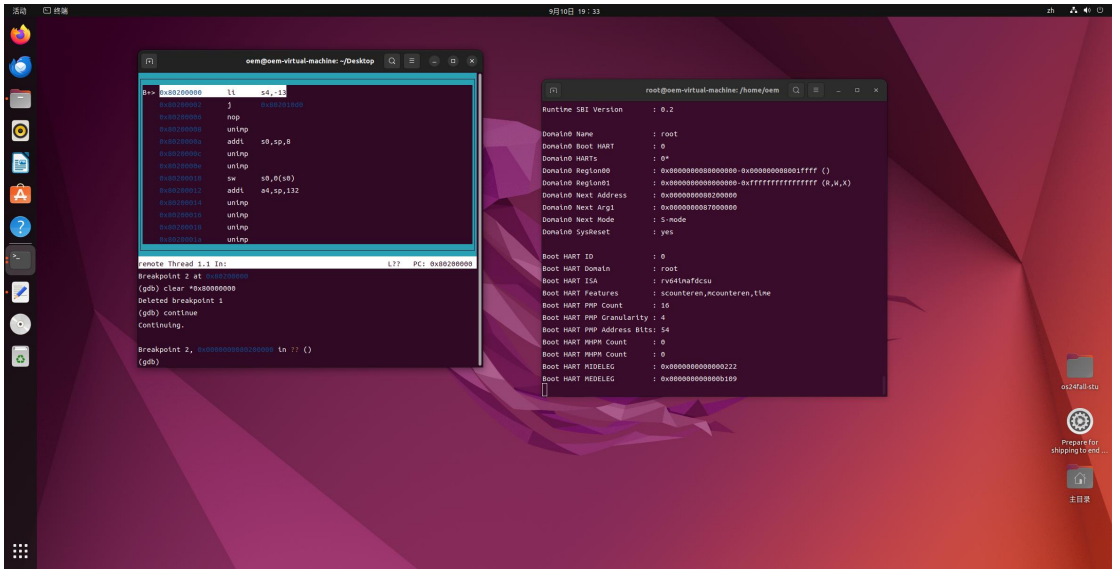
⑤清除 0x80000000 处的断点。

直接输入 clear *0x80000000 即可。



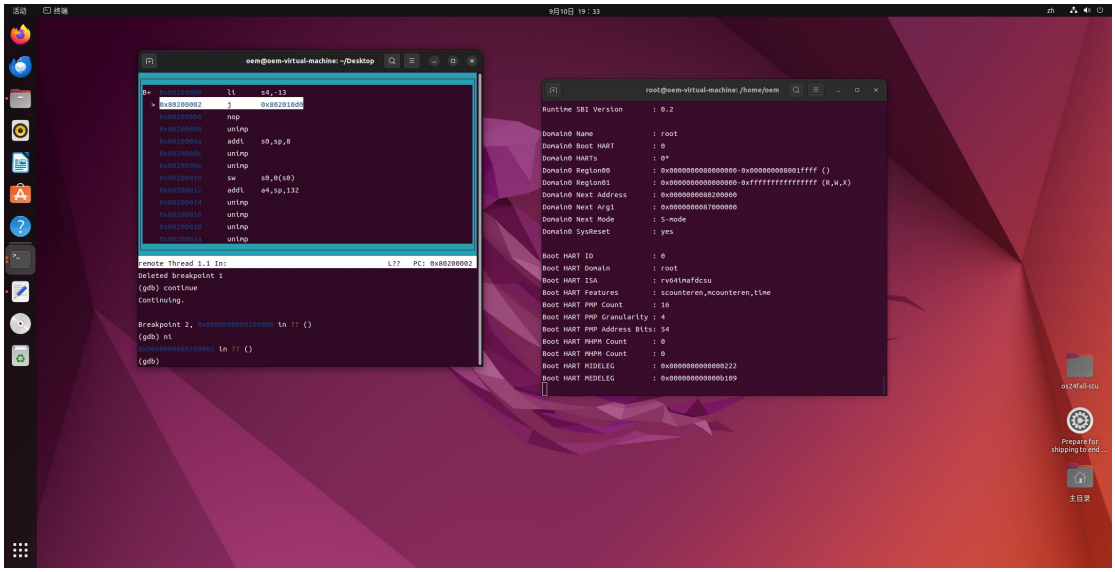
⑥继续运行直到触发 0x80200000 处的断点。

直接输入 continue 即可。



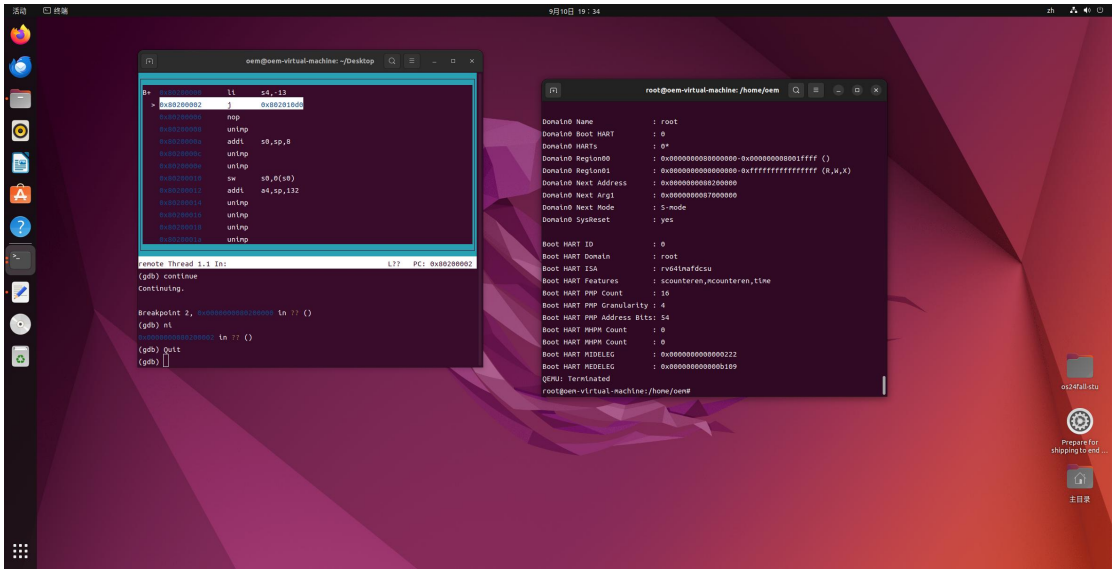
⑦单步调试一次。

直接输入 ni 即可。



⑧退出 QEMU。

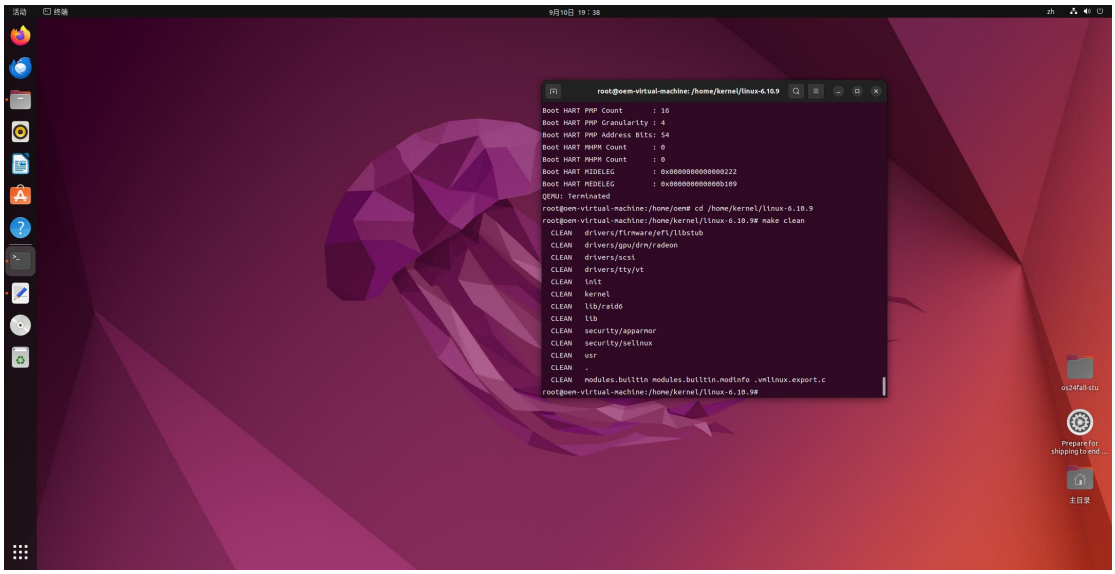
先在 gdb 中按 ctrl+C，再在 qemu 中按 ctrl+A 然后按 x。



4. 使用 make 工具清除 Linux 的构建产物

①进入编译文件夹。

②输入 make clean。



5. vmlinux 和 Image 的关系和区别是什么？

vmlinux 是未压缩的 Linux 内核镜像文件。它是内核在编译后生成的文件之一，包含了完整的内核代码、数据以及调试信息，通常是 ELF（Executable and Linkable Format）格式，这是一种标准的文件格式，包含了符号表和调试信息，使得开发人员能够进行调试和分析。由于 vmlinux 包含了完整的调试信息，它常用于调试目的和生成内核符号表。它不是直接用于引导的文件，而是用于开发和调试。

Image 是经过压缩的内核镜像文件，用于实际引导操作系统。它通常是经过压缩的、适合于引导加载程序（如 GRUB）或 bootloader（如 U-Boot）加载的文件。Image 文件通常采用 zImage、bzImage 或其他压缩格式。zImage 是一种较早的压缩格式，而 bzImage 是对其的改进，支持更大的内核镜像。不同的压缩格式可以影响引导时的内存使用效率。Image 是实际部署和引导系统时使用的内核文件。它通过引导程序被加载到内存中，然后执行以启

动操作系统。

五、附录

无。